ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОМУННИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

Лабораторная работа №5

«Многопоточное программирование»

Выполнил: студент гр. ИП-013

Копытина Т.А.

Проверил ассистент Кафедры ВС

Насонова А.О

Новосибирск 2022г.

Оглавление

ПОСТАНОВКА ЗАДАЧИ

1. Для программы умножения двух квадратных матриц DGEMM BLAS разработанной в задании 4 на языке C/C++ реализовать многопоточные вычисления. В потоках необходимо реализовать инициализацию массивов случайными числами типа double и равномерно распределить вычислительную нагрузку. Обеспечить возможность задавать размерность матриц и количество потоков при запуске программы. Многопоточность реализовать несколькими способами.

   1) С использованием библиотеки стандарта POSIX Threads.

   2) С использованием библиотеки стандарта OpenMP.

   3) * С использованием библиотеки Intel TBB.

   4) ** С использованием библиотеки стандарта MPI. Все матрицы помещаются в общей памяти одного вычислителя.

   5) *** С использованием технологий многопоточности для графических сопроцессоров (GPU) - CUDA/OpenCL/OpenGL/OpenACC.

2. Для всех способов организации многопоточности построить график зависимости коэффициента ускорения многопоточной программы от числа потоков для заданной размерностиматрицы, например, 5000, 10000 и 20000 элементов.

3. Определить оптимальное число потоков для вашего оборудования.

4. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

ОПИСАНИЕ ПРОГРАММЫ

- `void DGEMM` – функция поэлементного умножения матрицы.
- `void DGEMM_OPT1` – функция построчного умножения матрицы.
- `void DGEMM_OPT2` – функция блочного умножения матрицы.
- `void rand_mass` – функция заполнения матрицы случайными числами.
- `void* DGEMM_POSIX` – функция умножения матрицы с использованием библиотеки стандарта POSIX Threads.
- `void DGEMM_MP` – функция умножения матрицы с использованием библиотеки стандарта OpenMP.
- `double RandDouble` – функция генерации случайного числа типа `double`.
- `void* ForRand_POSIX` – функция заполнения матрицы случайными числами типа `double`.
- `pthread_attr_init` – инициализирует объект атрибута условия.
- `pthread_attr_setdetachstate` – устанавливает атрибут состояния отсоединения.
- `PTHREAD_CREATE_JOINABLE` - указывает, что будет создана подключаемая нить.
- `pthread_attr_setscope` - устанавливает атрибут области планирования в указанном объекте атрибута. Атрибут области планирования потока определяет, применяются ли решения о планировании потоков к потокам в данном процессе или ко всем потокам в масштабах всей системы.
- `PTHREAD_SCOPE_SYSTEM` - потоки планируются относительно всех потоков в системе.
- `void ForRand_MP` – функция заполнения матрицы случайными числами типа `double`.
- `void single_func` – функция для работы с матрицей и потоками с использованием библиотеки стандарта POSIX Threads.
- `void single_MP` - функция для работы с матрицей и потоками с использованием библиотеки стандарта OpenMP.

РЕЗУЛЬТАТ ПРОГРАММЫ

```
glass@glass-VirtualBox:~/NewLabACS/ABC/lab5$ ./main single opt0 5000 8
657.091
glass@glass-VirtualBox:~/NewLabACS/ABC/lab5$ ./main single opt0 5000 4
445.246
glass@glass-VirtualBox:~/NewLabACS/ABC/lab5$ ./main single opt0 5000 2
680.426
glass@glass-VirtualBox:~/NewLabACS/ABC/lab5$ ./main single opt0 10000 8
3286.412
```

Рисунок 1Обычный метод перемножения на матрице 5000x5000

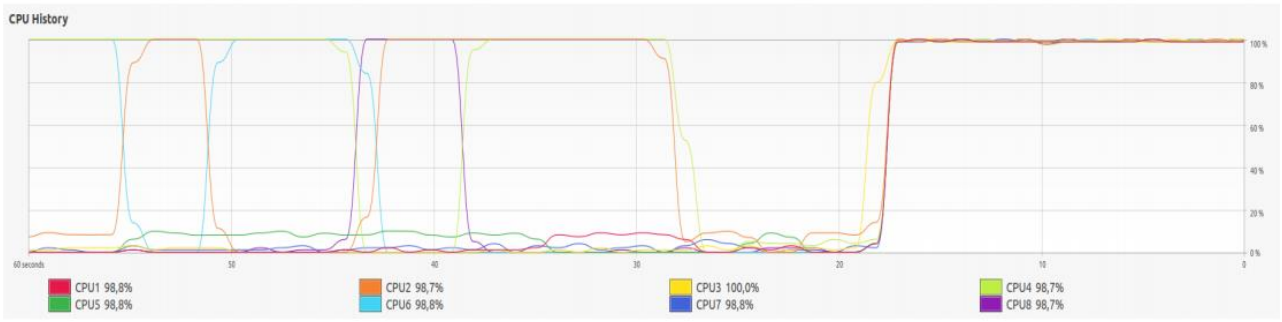


Рисунок 2 Загрузка процессора при запуске программы на 8 потоков

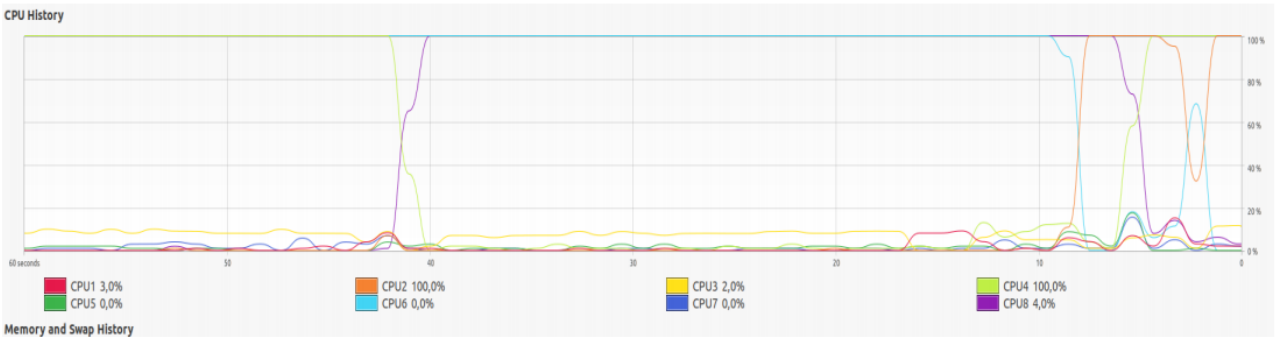
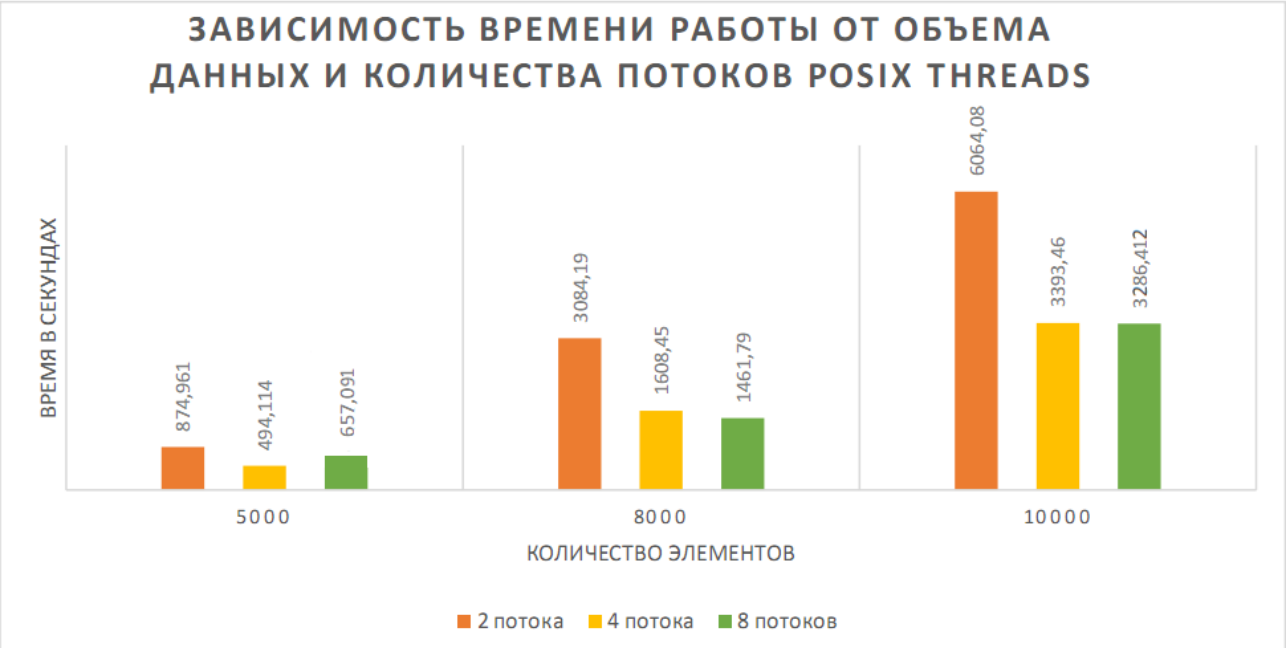
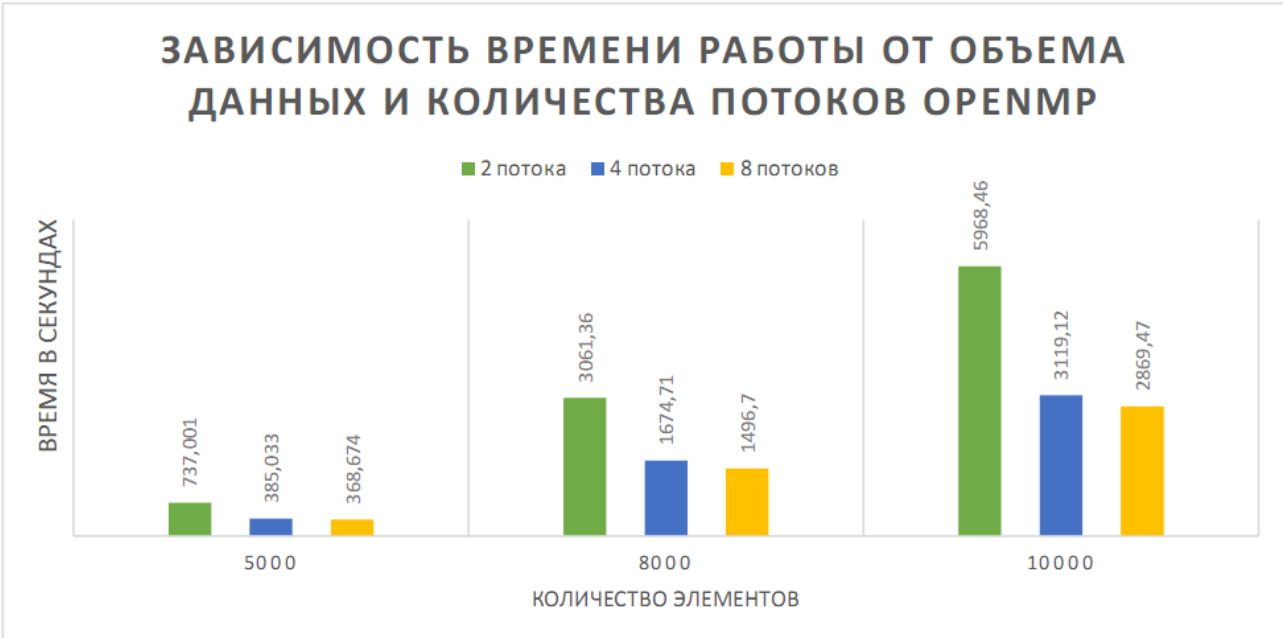


Рисунок 3 Загрузка процессора при запуске программы на 2 потока





По приложенным гистограммам видно, что самое наибольшее быстродействие достигается при количестве потоков равным 8. Также по графикам видно, что способ распараллеливания с помощью библиотеки OpenMP оказался эффективнее, чем Posix.

ThreadCount	1	2	3	4	5	6	7
posix	1.02091	2.00711	2.89022	3.32183	3.31405	3.22143	3.24602
open mp	0.976276	1.96523	2.83851	3.32581	3.33937	3.31037	3.32826

ThreadCount	1	2	3	4	5	6	7
posix	1,02091	2,00711	2,89022	3,32183	3,31405	3,22143	3,24602
open mp	0,976276	1,96523	2,83851	3,32581	3,33937	3,31037	3,32826

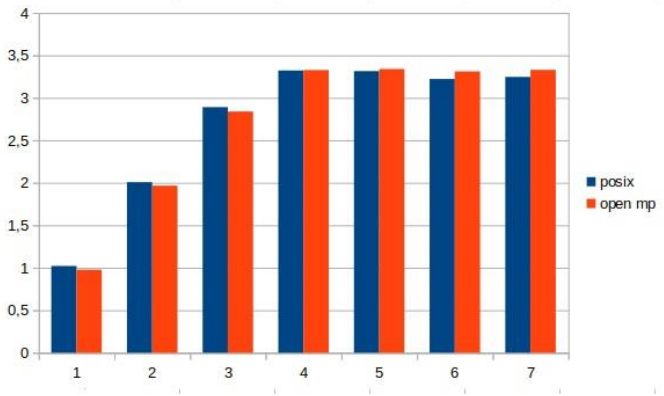


Рисунок 4. График ускорения

Чтобы построить график ускорения, мы делим вариант с несколькими потоками на обычное умножение. Например, считаем время с 2-мя, 3-мя и т.д. потоками при размере матрицы 5000x5000 и делим на время вычисления в случае с одним потоком.

ЛИСТИНГ

```
#include <bits/stdc++.h>
#include <omp.h>

using namespace std;

typedef struct {
    double** AdrMatr;
    int start;
    int interv;
    int N;
} ParamsRand;

typedef struct{
    double **Matrix_1;
    double **Matrix_2;
    double **Result;
    int start;
    int interv;
    int N;
} ParamsDgemm;

double RandDouble(double MinVal, double MaxVal)
{
    double d = (double)rand() / RAND_MAX;
    return MinVal + d * (MaxVal - MinVal);
}

void* ForRand_POSIX(void *data){
    ParamsRand *p = (ParamsRand*)data;
    double** Matrix = p->AdrMatr;
    int interv = p->interv;
    int N = p->N;
    const int start = (p->start!=N) ? p->start : 0;

    for(int i = start; i < (start+interv) && i < N; i++){
        for(int j = 0; j < N; ++j){
            Matrix[i][j] = RandDouble(1, 100);
        }
    }
    pthread_exit(NULL);
}

void RandMatrix_POSIX(double **Matrix,int N,int sum_thread){
    pthread_t* thread = new pthread_t[sum_thread];
    pthread_attr_t attr;
    pthread_attr_init( &attr );
    pthread_attr_setdetachstate( &attr, PTHREAD_CREATE_JOINABLE );
    pthread_attr_setscope( &attr, PTHREAD_SCOPE_SYSTEM);

    int interval = N / sum_thread;

    ParamsRand par;
    par.AdrMatr = Matrix;
    par.interv = interval;
    par.N = N;
    par.start = 0;
```

```
    for(int i = 0; i < sum_thread; i++){
        pthread_create(&thread[i],&attr,ForRand_POSIX,(void*)&par);
        par.start += interval;
    }

    for(int i = 0; i < sum_thread; i++) pthread_join(thread[i],NULL);
}

void ForRand_MP(double **a, double **b, long long n, int potoks){
    #pragma omp parallel num_threads(potoks)
    {
        int threadid = omp_get_thread_num();
        int items_per_thread = n / potoks;
        int lb = threadid * items_per_thread;
        int ub = (threadid == potoks - 1) ? (n - 1) : (lb +
                                                       items_per_thread -
1);
        for (int i = lb; i < ub; i++) {
            for (int j = 0; j < n; ++j) {
                a[i][j] = RandDouble(1, 100);
                b[i][j] = RandDouble(1, 100);
            }
        }
    }
}

void* DGEMM_POSIX(void *data) {
    ParamsDgemm *p = (ParamsDgemm*)data;
    double** a = p->Matrix_1;
    double** b = p->Matrix_2;
    double** c = p->Result;
    int interv = p->interv;
    int N = p->N;
    const int start = (p->start!=N) ? p->start : 0;

    for (long long i = start; i < start + interv; i++) {
        for (long long j = 0; j < N; j++) {
            for (long long k = 0; k < N; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

void DGEMM_MP(double **a, double **b, double **c, long long N, int
potoks) {
    #pragma omp parallel num_threads(potoks)
    {
        int threadid = omp_get_thread_num();
        int items_per_thread = N / potoks;
        int lb = threadid * items_per_thread;
        int ub = (threadid == potoks - 1) ? (N - 1) : (lb +
                                                       items_per_thread -
1);

        for (long long i = lb; i < ub; i++) {
            for (long long j = 0; j < N; j++) {
                for (long long k = 0; k < N; k++) {
```

```
                    c[i][j] += a[i][k] * b[k][j];
                }
            }
        }
    }
}


void single_func(string multiply, int n, int potoks) {

    if (multiply == "opt0") {
        double t = omp_get_wtime();
        double **a, **b, **c;
        a = new double *[n];
        b = new double *[n];
        c = new double *[n];

        for (long long i = 0; i < n; i++) {
            a[i] = new double[n];
            b[i] = new double[n];
            c[i] = new double[n];
            for (long long j = 0; j < n; j++) {
                a[i][j] = 0;
                b[i][j] = 0;
                c[i][j] = 0;
            }
        }

        RandMatrix_POSIX(a, n, potoks);
        RandMatrix_POSIX(b, n, potoks);

        pthread_t* thread = new pthread_t[potoks];
        pthread_attr_t attr;
        pthread_attr_init( &attr );
        pthread_attr_setdetachstate( &attr, PTHREAD_CREATE_JOINABLE );
        pthread_attr_setscope( &attr, PTHREAD_SCOPE_SYSTEM );

        int interval = n / potoks;

        ParamsDgemm param;
        param.Matrix_1 = a;
        param.Matrix_2 = b;
        param.Result = c;
        param.start = 0;
        param.interv = interval;
        param.N = n;

        for(int i = 0; i < potoks; i++){
            pthread_create(&thread[i],&attr,DGEMM_POSIX,(void*)&param);
            param.start += interval;
        }
        for(int i = 0; i < potoks; i++) pthread_join(thread[i],NULL);
        t = omp_get_wtime() - t;
        cout << t << endl;
         for (int i = 0; i < n; i++) {
                delete[]a[i];
                delete[]b[i];
                delete[]c[i];
```

```
            }
            delete[]a;
            delete[]b;
            delete[]c;
    }
}

void single_MP(string multiply, int n, int potoks) {

    if (multiply == "opt0") {
        double t = omp_get_wtime();
        double **a, **b, **c;
        a = new double *[n];
        b = new double *[n];
        c = new double *[n];

        for (long long i = 0; i < n; i++) {
            a[i] = new double[n];
            b[i] = new double[n];
            c[i] = new double[n];
            for (long long j = 0; j < n; j++) {
                a[i][j] = 0;
                b[i][j] = 0;
                c[i][j] = 0;
            }
        }

        ForRand_MP(a, b, n, potoks);

        DGEMM_MP(a, b, c, n, potoks);
        t = omp_get_wtime() - t;
        cout << t << endl;
        for (int i = 0; i < n; i++) {
                delete[]a[i];
                delete[]b[i];
                delete[]c[i];
            }
            delete[]a;
            delete[]b;
            delete[]c;
    }
}

int main(int argc, char** argv) {
    if(argc < 3){
        cout << "Слишком мало аргументов";
        return 1;
    } else if((string)argv[1] == "single"){
        single_func((string)argv[2], atoi(argv[3]), atoi(argv[4]));
    } else if((string)argv[1] == "single_mp"){
        single_MP((string)argv[2], atoi(argv[3]), atoi(argv[4]));
    }
}
```